Chaotic-Identity Maps for Robustness Estimation of Exascale Computations

Nagi Rao

(Nageswara S. V. Rao)

Oak Ridge National Laboratory

IEEE Workshop on Fault Tolerance for HPC at Extreme Scale

June 25, 2012, Boston, MA

Research Sponsored by

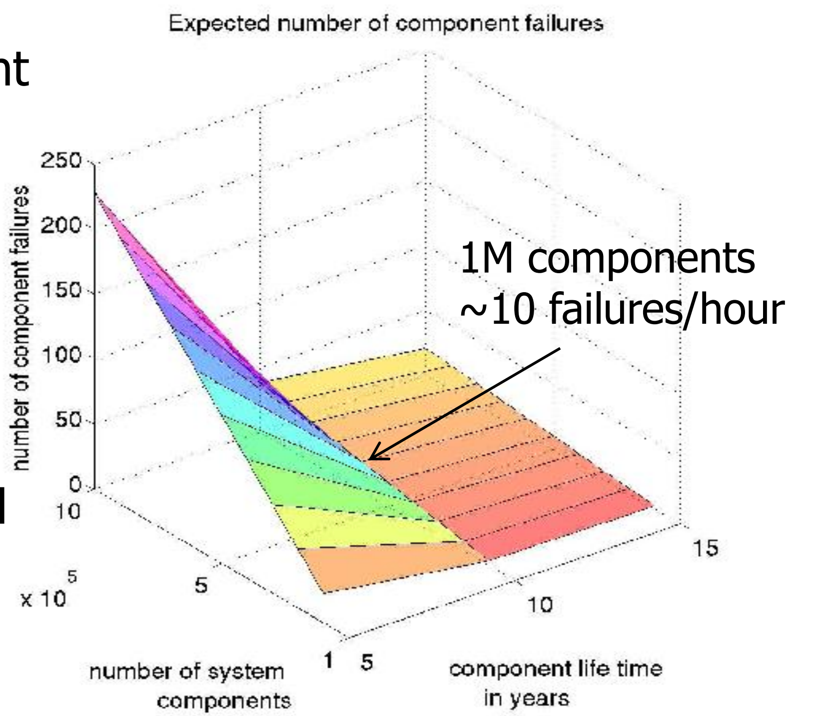
Applied Mathematics Program, U.S. Department of Energy

Outline

1. Introduction
2. Chaotic-Identity Maps
3. Diagnosis Pipelines
4. Confidence Estimates
5. Simulation Results
6. Conclusions

Inherent Failures in Exascale Computing Systems

- Exascale computing systems are expected to have processor cores and other components in the numbers of millions.
  - components with expected life-span of ten years
    - ~100k hours/component = 10 failures among 1M components
  - codes that run for a few hours likely experience failures of several components.
- Failure rates limit the effectiveness of current check-pointing:
  - run-times could be of the order of several hours for exascale systems
  - transient silent errors may lead to erroneous computations
- Failures will be integral part of exascale computations – must be explicitly accounted
  - code outputs must be quantified with confidence estimates
    - specific to system failure profile
    - justifiable by measurements



Related Areas

- Foundational works:
  - von Neumann studied (in 1950s) mathematical aspects of achieving reliable computations over systems with unreliable components
  - subsequent reliability improvements in computing systems, perhaps, led to such studies not being extensively continued
- Deployed systems: computing systems in satellites
  - deployed over past decades - enhanced with Software-Implemented Hardware Fault Tolerance (SIHFT) methods to counteract errors due to radiation in space environments.

But, exacale computations present new challenges

- sheer size and system complexity makes dynamic profiling of the failures and robustness complicated
- computation becomes inherently probabilistic:
  - for most applications, 100% guarantee of robustness against failures in not possible
  - requires confidence measures for code outputs – running to completion is not sufficient

System Profiling and Application Tracing

System Diagnosis and Profiling:

- Executed at the beginning for an initial system profile
  - repeated periodically or triggered by failure events.
- Typically, all system resources are devoted for initial profiling

- Our method:
  - execute diagnosis modules customized to static and silent failures in processing nodes, memory units and interconnects
  - generate robustness estimates from outputs of diagnosis modules.

Application Tracing:

- diagnosis modules are strategically inserted into application codes
  - during compilation or preprocessing
- confidence measures are estimated for their outputs.

Basic idea: execution paths of these tracer codes “follow” along the same components as the application codes:

- processing nodes, memory elements and interconnect links,

Very important case: no detected failures lead to higher confidence for application codes – detection is only a part of our goal

Our Approach

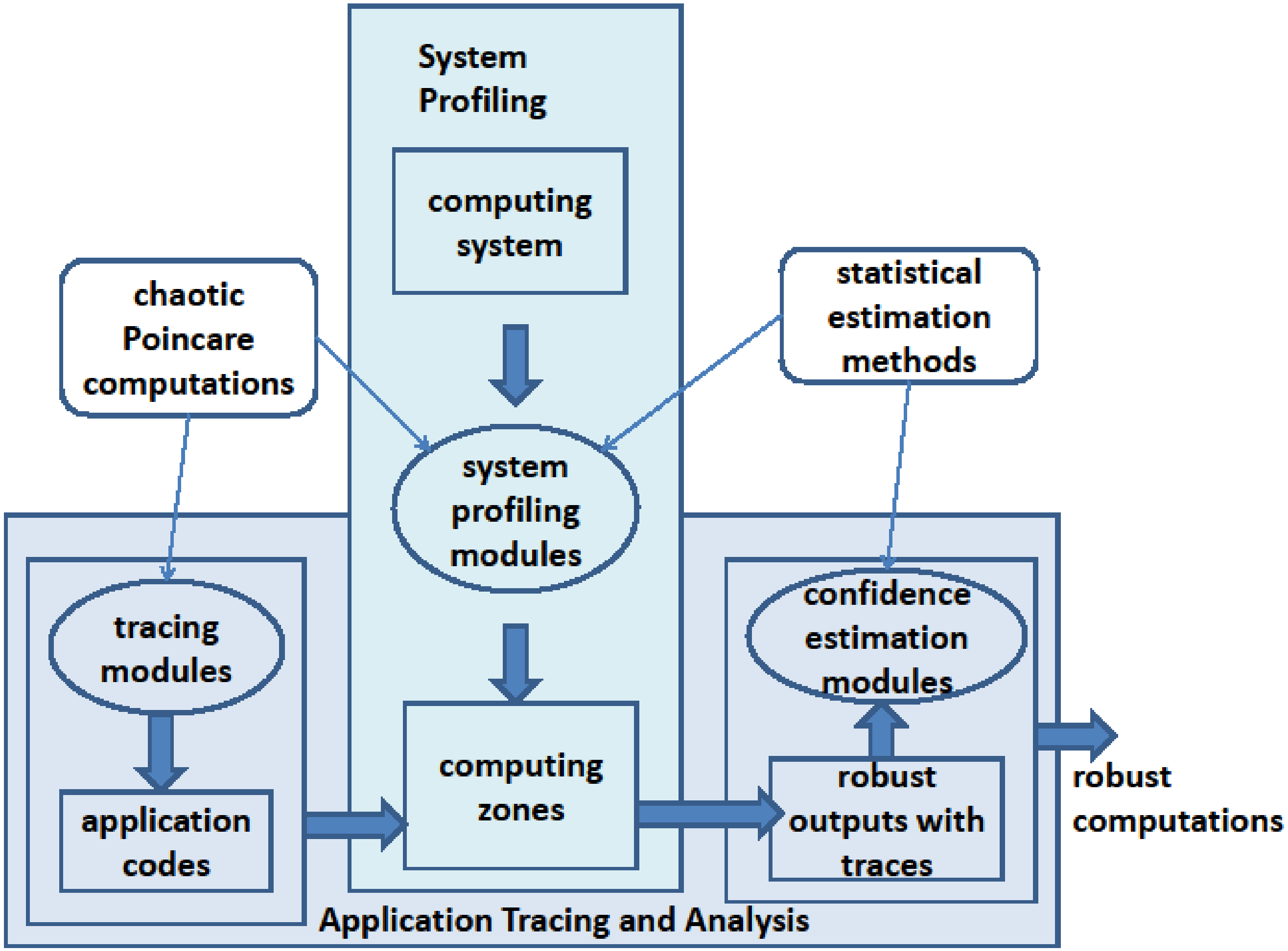
Our approach: synthesis of methods from fault diagnosis, chaotic Poincare maps, and statistical estimation:

a) **Diagnosis methods:** identify computation errors due to component failures, in arithmetic and logic unit (ALU), memory and cross-connect, by strategically guiding the execution paths:
   i. system diagnosis pipelines
   ii. application traces

b) **Poincare maps** amplify effects of component failures making them quickly detectable,

c) **Statistical estimation** methods process data from execution traces to generate
   i. system robustness profiles
   ii. confidence estimates for applications

Framework for System Profiling and Application Tracing



System profiles can be used to identify computing zones
Applications can be executed in suitable zones and traced to generate confidence estimates

Chaotic Poincare maps

Poincare Map: $M:\Re^d \to \Re^d$

$$X_{i+1} = M\left(X_i\right)$$

Trajectory

$$X_0, X_1, X_2, \cdots$$

Examples:

logistic map: $X \in [0,1]$

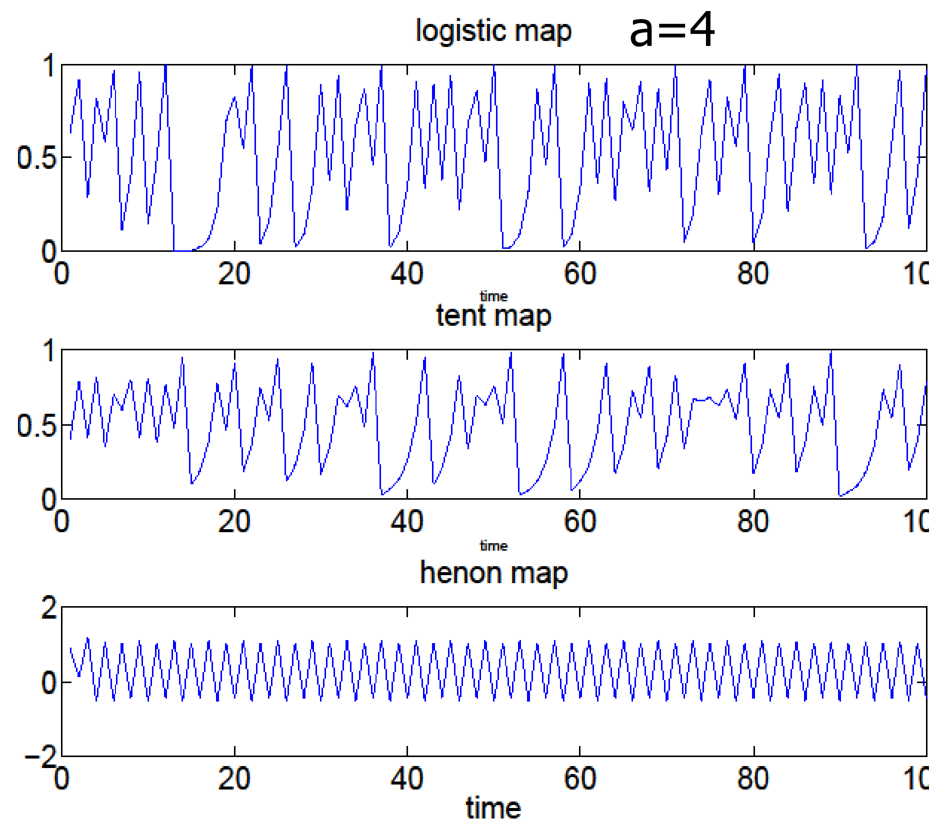
$$M_{L_a}\left(X\right) = aX\left(1-X\right)$$

tent map: $X \in [0,1]$

$$M_T\left(X\right) = \begin{cases} 2X & if \quad X \le 1/2 \\ 2\left(1-X\right) & if \quad X > 1/2 \end{cases}$$

Hennon map

$$M_H\left(X,Y\right) = \left(a - X^2 + bY, X\right)$$



Simple computations generate seemingly complex trajectories

Chaotic maps amplify state errors

Chaotic trajectories: $X_0, X_1, X_2, \cdots$ is chaotic if

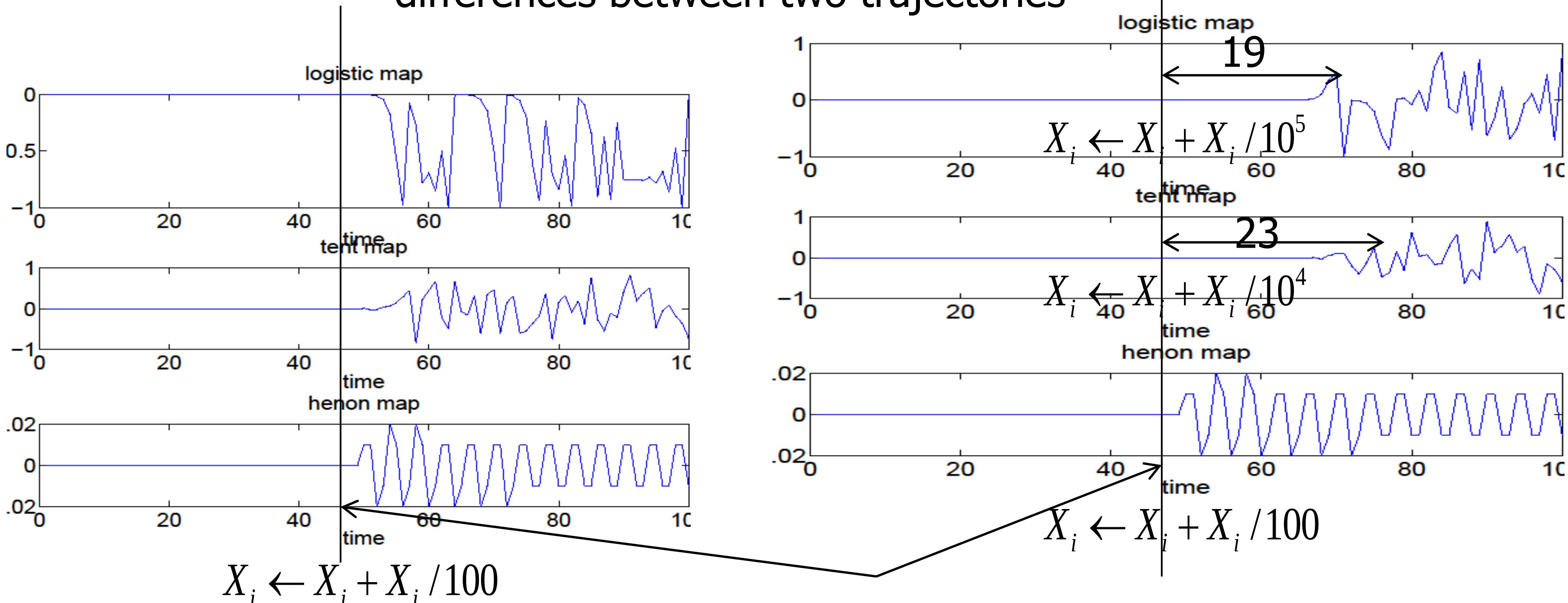
(i) it is not asymptotically periodic, and

(ii) Lyapunov exponent is positive

$$\mathrm{L}_M = \ln\left|\frac{dM}{dX}\right| > 0$$

Key Property: Extreme sensitivity to states: small differences in states lead to rapidly divergent trajectories

differences between two trajectories



one of the states corrupted at t=50

Poincare maps for fault detection

Poincare maps computed in parallel at different nodes: fault at one will lead to quick divergence of the outputs, depending on:

- **Type of faults**: Wide range of faults in
  - arithmetic and logical operations
  - registers and memory

  but are limited to those in operations used by M(.)

- **Poincare map properties**: Computation of M(.)
  - sensitive to errors
    - in constituent operations, and
    - mechanisms used in storing and updating the states
  - rate of divergence and its detectability depends on the Lyapunov exponent
    - generally, larger Lyapunov exponent values lead to quicker divergence
    - for tent map, $\mathrm{L}_M = \ln 2 > 0$ except at *X=1/2*

Side Note: Codes with known outputs are routinely used for diagnosis of computing systems – Poincare maps are among the least complex

Chaotic-Identity Map

Poincare map amplifies errors in operations used in its own computation

Chaotic-Identity Map:

$$\tilde{X}_i \leftarrow I_D\left(X_i\right)$$
$$X_{i+1} \leftarrow M\left(\tilde{X}_i\right)$$

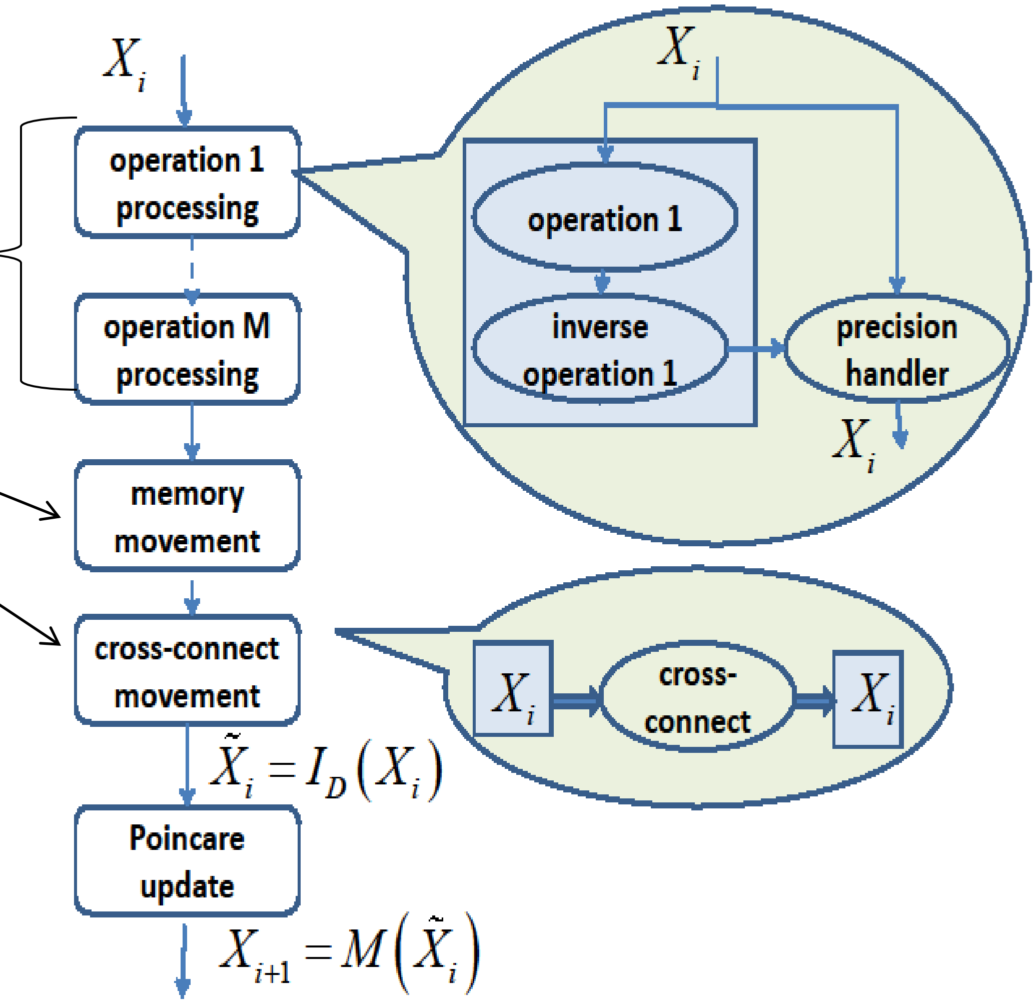
Execution routed through

- computing operations
- memory locations
- interconnect links

to capture errors in them

Output $I_D\left(X_i\right)$ is identical to X_i if there are no faults

It catches errors in specified operations – instructions, sub-routines, libraries



Chaotic-Identity Map

Chaotic-Identity map (CI-map) augments Poincare computation :

- **Operation-Inverse Pairs:** each update step with a sequence of pairs each consisting of an operation and its inverse. Choice based on instruction sets of CPU and GPU, sub-routines, libraries
  - complement operations used by Poincare map operations.

  Application of a pair of operations gives back the original operand

  - error in either would be amplified by subsequent Poincare updates

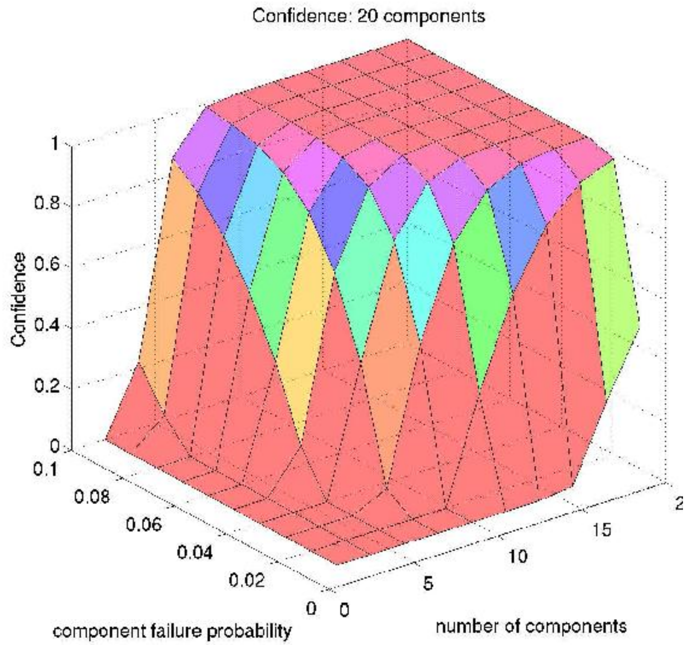
- **State Movement Operations**: move state variable
  - among the memory elements and/or
  - across the interconnects, in each step

  before applying *M(.)*

  Capture errors in memory and transmission across interconnect
  - memory-to-memory transfers can be achieved by several means:
    - -additional variables in “shared” memory, explicit MPI calls
  - application tracing: movements reflect execution paths of the application - tracer codes are called from within them.

Confidence Estimates

Outputs of CI-maps are used to generate confidence measures for executions, particularly if no failures are detected



$I_D(.); M(.)$ executed at rate R_P
- once every $1/R_P$ seconds

P_{1/R_P} probability of node failure during $1/R_P$ sec

Under statistical independence
probability of failure during N_P executions

$$1-\left(1-P_{1/R_P}\right)^{N_P}$$

Confidence: $C(\alpha, N_P)$
that node failure probability is less than α

If no failures are detected in N_P executions

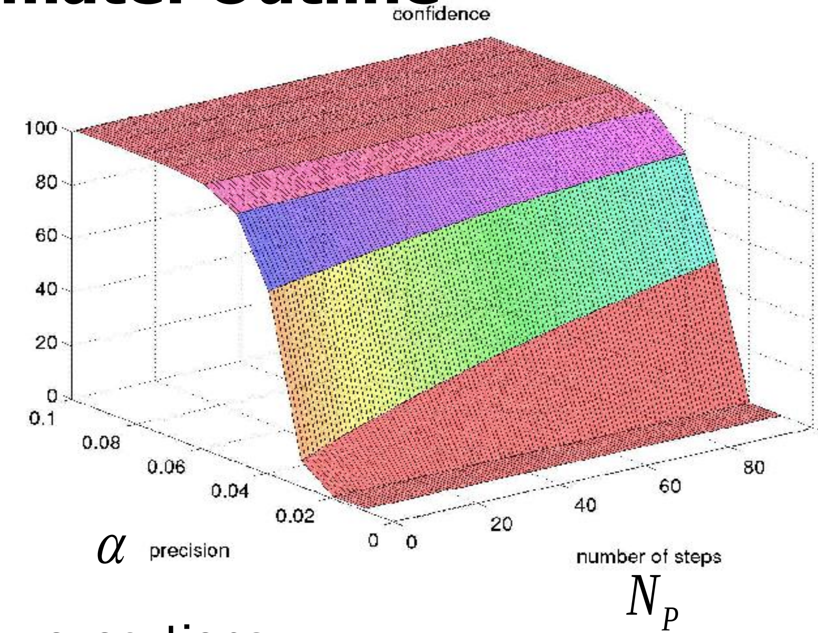
$$C(\alpha, N_P) = P\left\{P_{1/R_P} < \alpha\right\} > 1 - 2^{-2\left[1-(1-\alpha)^{N_P}\right]^2 N_P}$$

Derivation of Confidence Estimate: Outline

By Hoeffding's Inequality we have

$$P\left\{\left|1-\left(1-P_{1/R_P}\right)^{N_P}\right|>\in\right\}<2^{-2\in^2 N_P}$$

$$P\left\{P_{1/R_P}<\alpha\right\}>1-2^{-2\left[1-\left(1-\alpha\right)^{N_P}\right]^2 N_P}$$



General Confidence Estimate:

If failures are detected in $\hat{P}_E$ fraction of N_P executions

General confidence estimate:

$$C\left(\alpha, N_P\right)=P\left\{P_{1/R_P}<\alpha\right\}>1-2^{-2\left[1-\left(1-\alpha\right)^{N_P}-\hat{P}_E\right]^2 N_P}$$

Derivation: By Hoeffding's Inequality we have

$$P\left\{\left|\left(1-P_{1/R_P}\right)^{N_P}-\hat{P}_E\right|>\in\right\}<2^{-2\in^2 N_P}$$

$$P\left\{\left|P_{1/R_P}-\hat{P}_E\right|<\beta\right\}>1-2^{-2\left[1-\left(1-\beta\right)^{N_P}\right]^2 N_P}$$

Generic CI-Map

Generic CI-map computation $X_{i+1} \leftarrow_{L_{j,k}} I_{D:P_J}\left(X_i\right)$

$I_{D:P_j}\left(X_i\right)$ is computed on computing node P_j

output $\left(i, X_i\right)$ is sent to the computing node P_k

Trajectory generated by *n* Poincare map computations on node P

$$\left[n, X_0, X_n\right]_P$$

Output of computation triplet $\left(n, X_0, X_n\right)$

PCC-Chains

Poincare Computing and Communication chain utilizes computations n processing nodes

$$\mathrm{P} = \left\{ P_0, P_1, \cdots, P_{n-1} \right\}$$

connected over interconnect such that

$I_{D:P_i}\left(X_i\right)$ is computed on P_i and sent to P_{i+1} over interconnect link

Output of this chain $\left(n, M_{P_{n-1}}\left(X_{n-1}\right)\right)_{\mathrm{P}}$

computed in time $n\left(T_M + T_I\right)$

T_M :cost of computing
T_I :cost of communicating over interconnect

Pipelines of PCC-Chains

Compose a Pipelined Chains of Chaotic PCC maps (PCC^2-map) by using PCC-chains such that:

$I_{D:P_i}\left(X_{i+k}^{k}\right)$ of *k*-th chain

- computed on P_i at time $i + k$ and
- sent to P_{i+1} over interconnect link
- Example: computation sequence at P_0 is: $I_{D:P_0}\left(X_0^0\right), I_{D:P_0}\left(X_1^1\right), I_{D:P_0}\left(X_2^2\right), \cdots$

Computed in time: $(n+k)\left(T_M + T_I\right)$ in parallel

Confidence bound:

$$P\left\{P_{\tau_C} < \alpha\right\} > 1 - 2^{-2\left[1-(1-\alpha)^{n+k} - \hat{P}_{\tau_C}\right]^2 N_P}$$

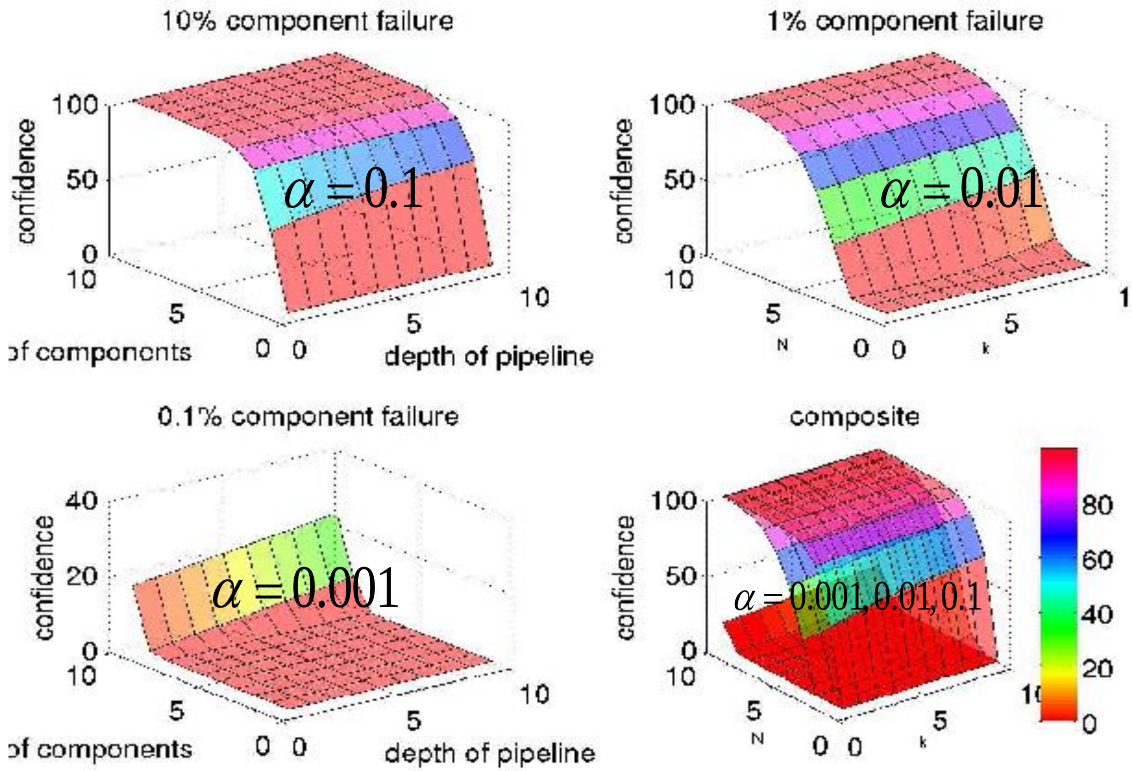
A pipeline with n_P chains and node-periodicity T_P uses consecutive block of nodes:

- chains sweep across all *N* nodes
- for full pipeline $n_P = T_P$

Confidence Estimates

k = 10, n = 10; no detected errors

$\alpha = 0.001, 0.01, 0.1$



With no detected faults, higher confidences with:

(a) deeper pipelines
(b) more components
(c) lower precisions

Certain confidence levels can only be achieved with "deep enough" pipelines

Simulation Results

We simulate three types of errors:

i. ALU errors corrupt state by a multiplier
   - bit flip to 1 in ALU registers

ii. memory errors clamp state to a fixed value
   - stuck-at fault in RAM

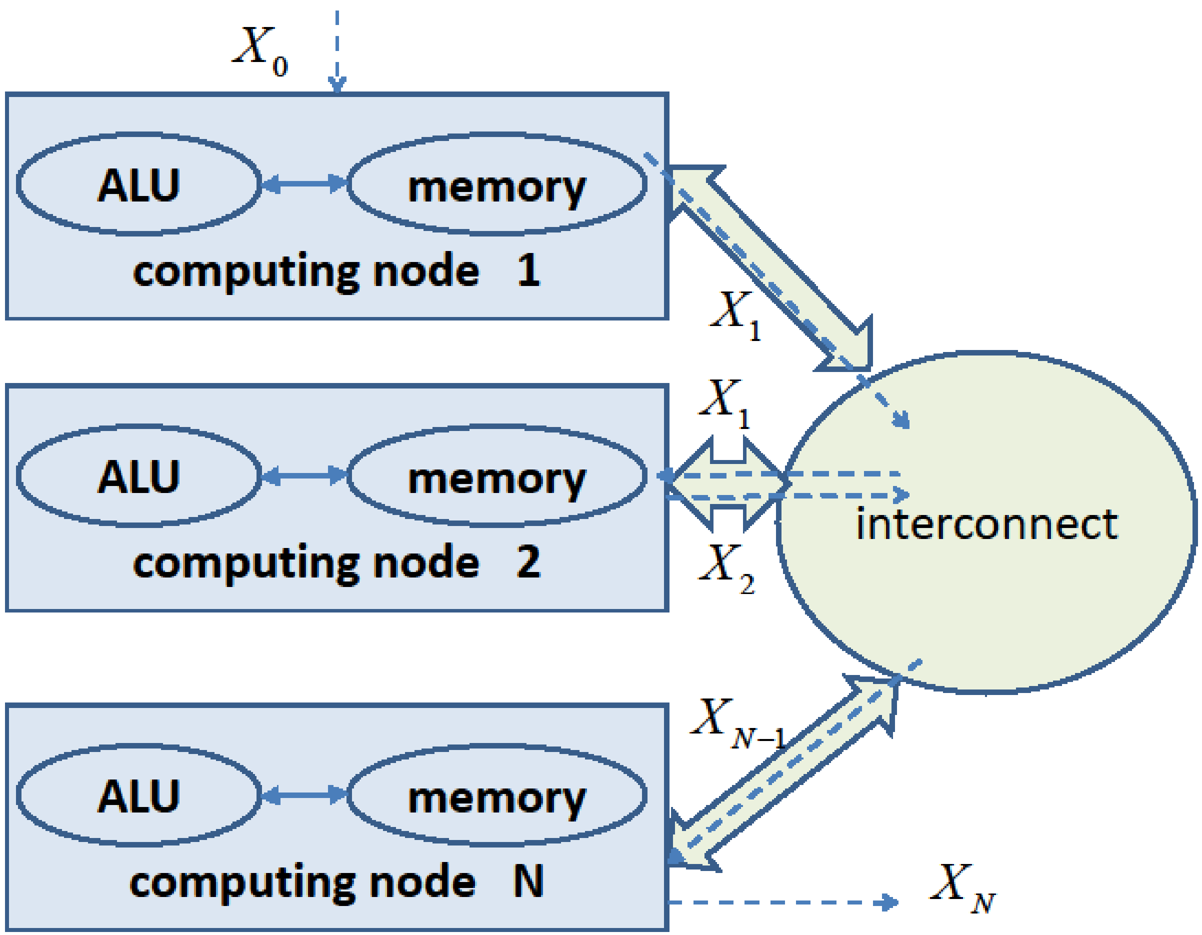
iii. cross-connect errors modify state by a multiplier.
   - link transmission error

Nodes transition to a faulty mode with probability *p*, and once transitioned

- errors type (i) and (ii) are permanent,
- error type (iii) lasts only for a single time step

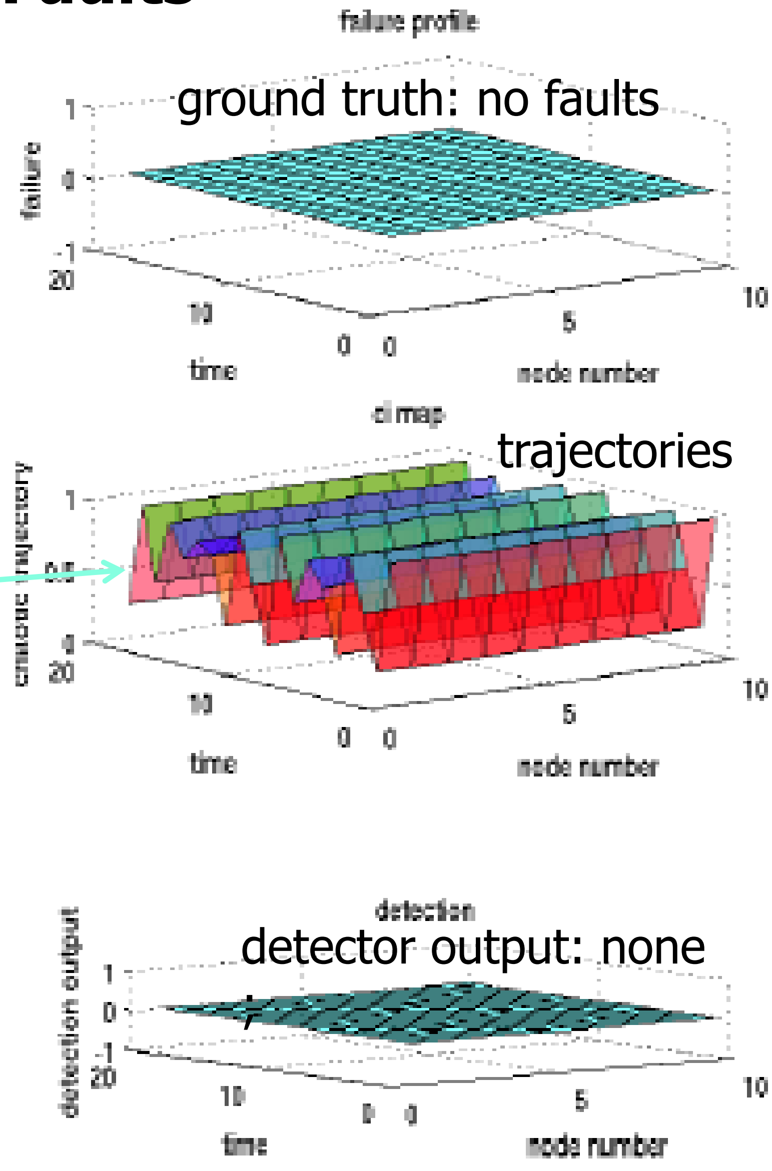
Simulation Abstraction

Computation of PCC-chain is routed through nodes via interconnect



Simulation Results: No Faults

Case of no faults:
10-node pipeline of depth *k = 10*
- none are detected
- all chaotic time traces are identical across nodes

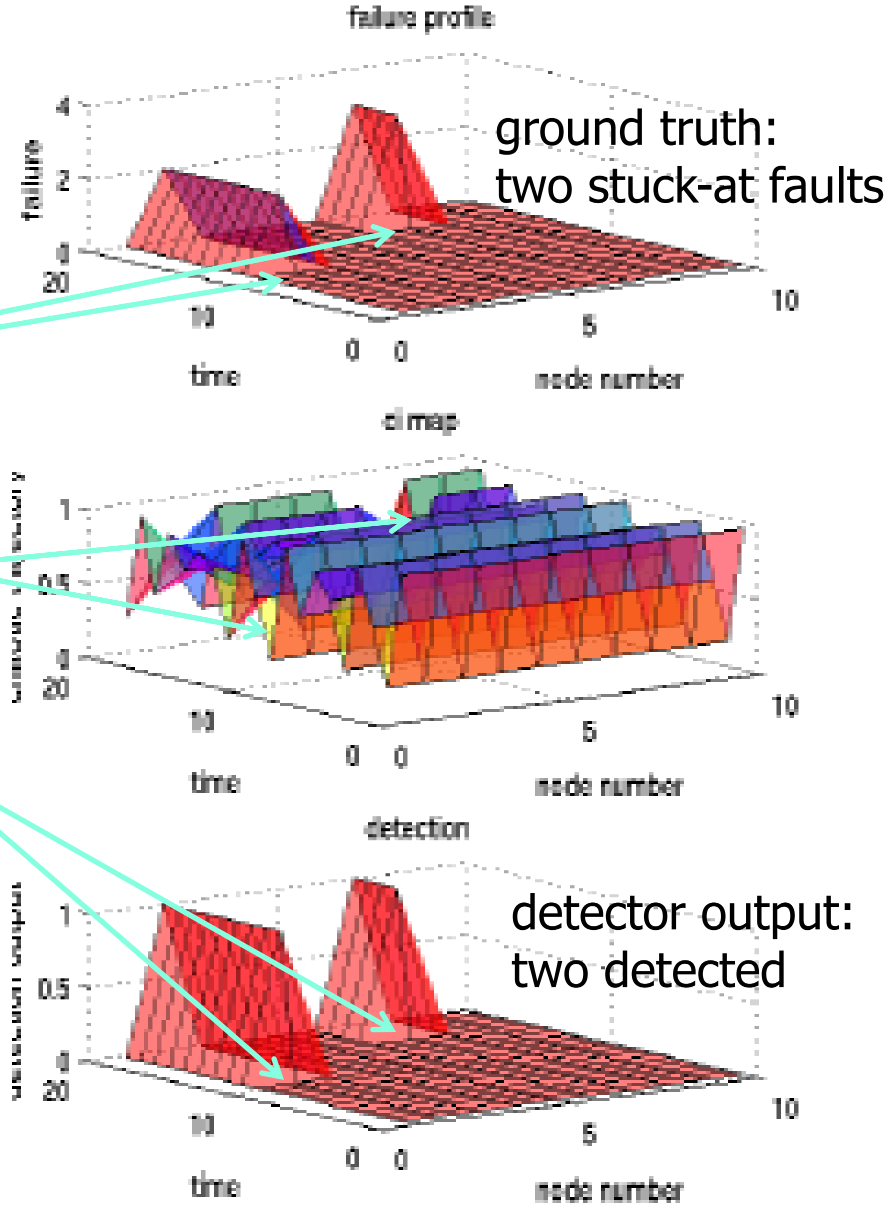


(a) no failures

Simulation Results

Stuck-at faults:
•full pipeline, spanning all 10 nodes
•trajectories disrupted by faulty nodes

•detection within one time step



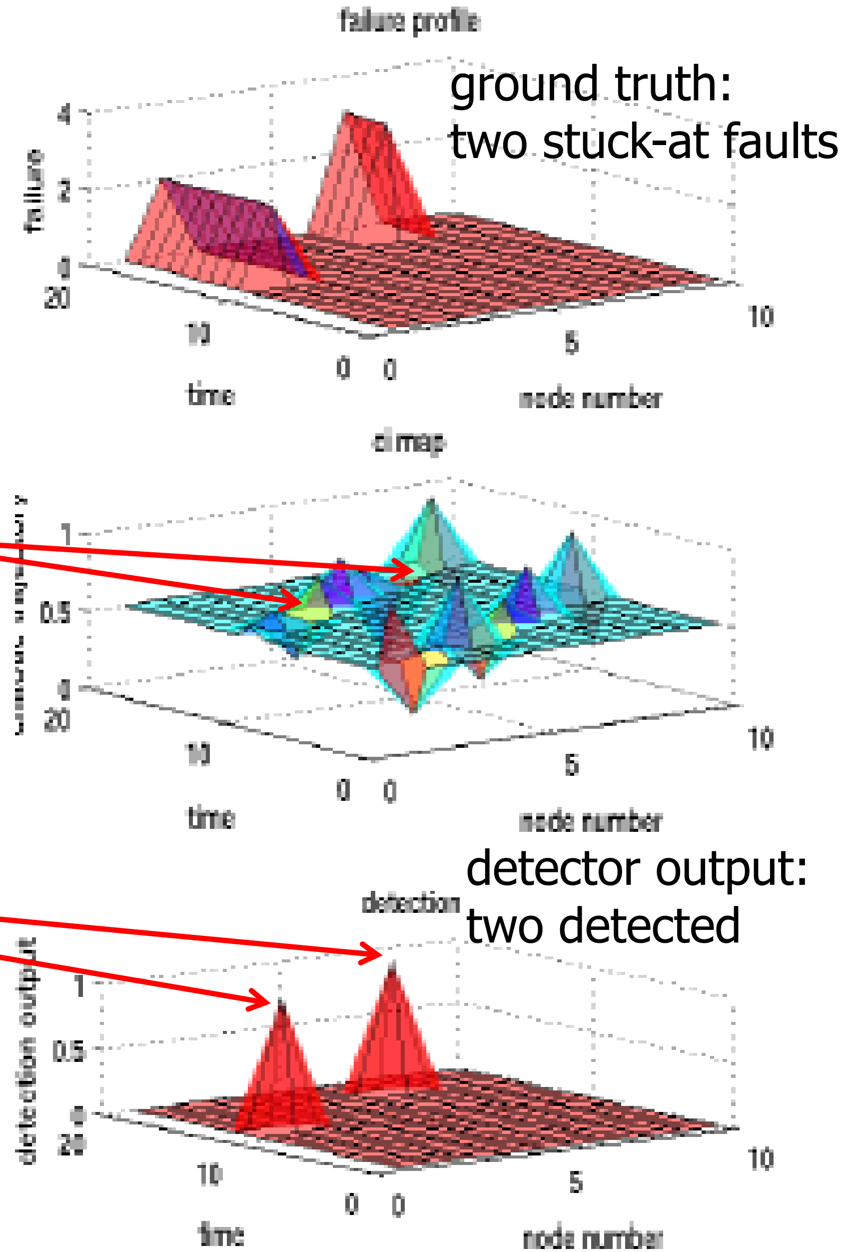
(b) full pipeline for stuck-at failures

Simulation Results

Pipeline of single chain
- executed by one node at time
- chain “sweeps” across nodes in time

Both faults are detected:
- detection delayed until the chain reaches faulty node

The total computational cost:
- 1/10 of the case (b)
- detection achieved, albeit delayed by few time steps



ground truth: two stuck-at faults

detector output: two detected

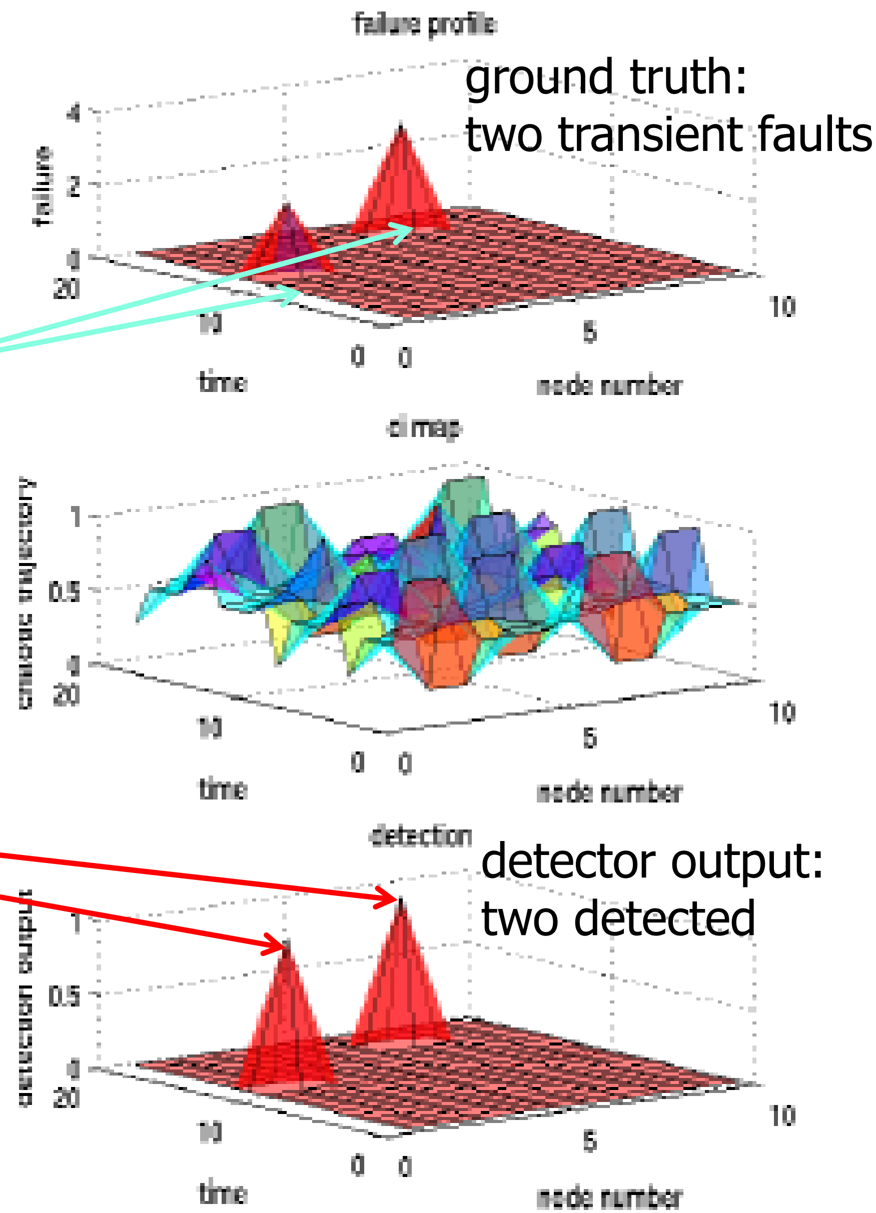
c) sparse pipeline for stuck-at failures

Simulation Results

Transient fault in interconnect payload lasted for one time unit

Full pipeline spanning all nodes will detect such failure

Pipeline of two chains with periodicity of 5 nodes is able to detect



(d) transient failure

Simulation System

Simulations on 48-core Linux workstation: 2.23GHz AMD Opteron processors

Computation on a single processor core and delay of 10 micro seconds to simulate the latency of interconnect.

- $N = 500{,}000$ nodes: runtimes under 2 seconds for
  - logistic map and a pair of reciprocal operations (5 operations for CI-map).

First-order approximation: for CI-map

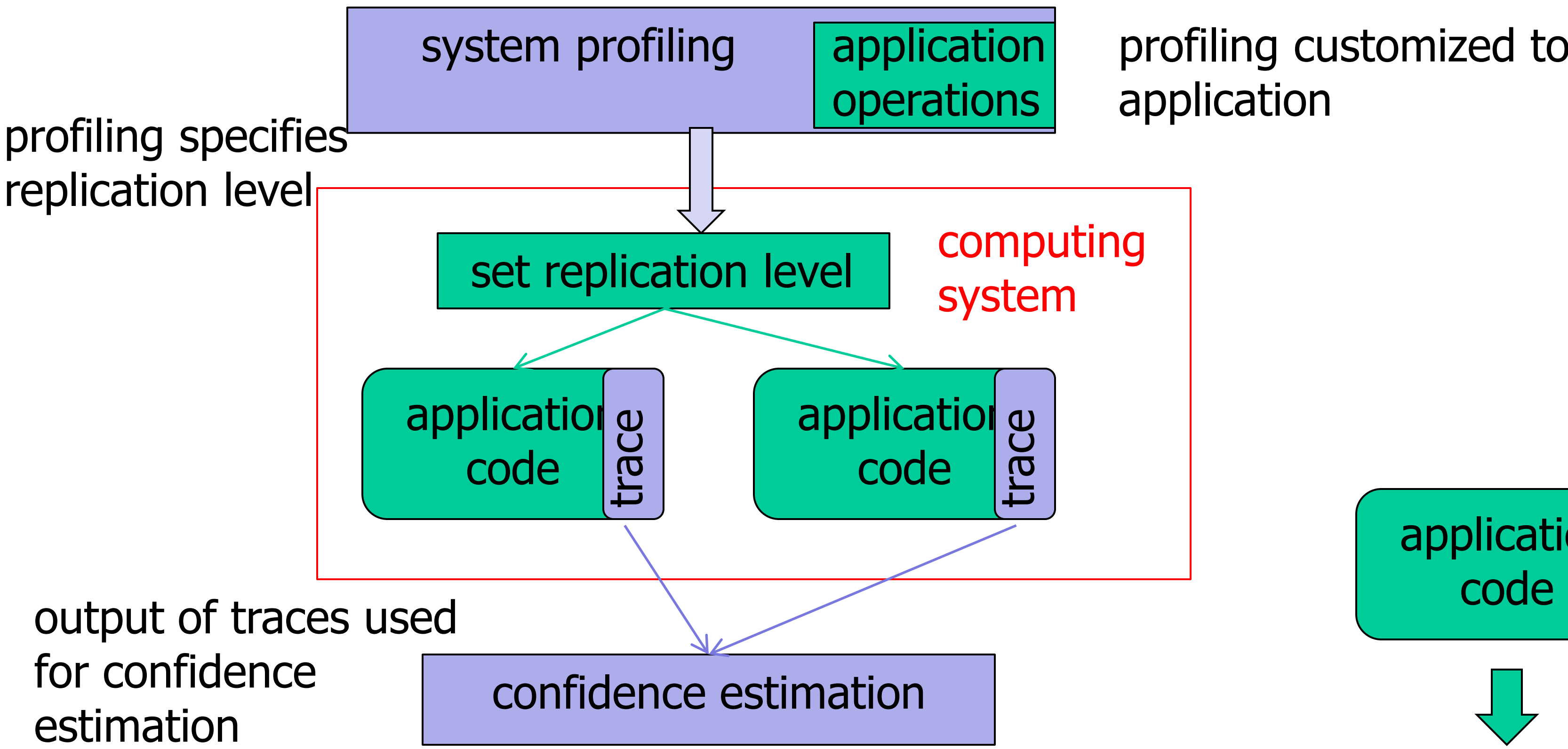
- 10 operations each with 10 micro seconds execution time, and
- interconnect with 10 microsecond latency

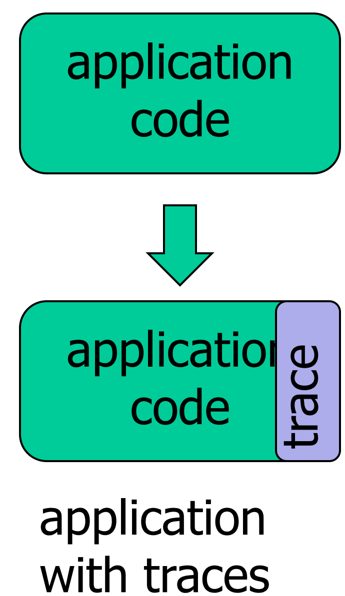
pipeline execution time is 11 seconds for $N=100{,}000$

All chains of PCC^2-map are computed in parallel

- execution time scales linearly in N
- under 2 minutes for million computing nodes

Replicated Application Execution





- Our approach “informs” application and quantifies its output
- Check-pointing can be similarly “informed” – failure rate estimates specify check-pointing frequency

Conclusions

Our approach

(i) utilizes light-weight computations based on chaotic and identity maps to detect certain classes of errors in computations, and

(ii) estimates system robustness and confidences of computations

We illustrated the concepts using simulation examples.

This approach is suitable for exascale systems:

(a) low computational requirements

(b) linear scaling of the execution time

both for system profiling and application tracing

Future Work:

- These results are only a very first step
- More analysis and simulations needed
  - understand and quantify classes of errors detected by a given set of Poincare and identity maps
- Statistical estimates are only first-order approximations:
  - further research required to handle correlated failures.

Thank you